JavaScript Classes

🚀 Everything about JS Classes in just **20 pages**

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Agenda

- Classes
- Creating Classes
- Static Methods
- Adding Methods
- Public Fields
- Private Fields
- Subclasses
- Delegation Over Inheritance

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Classes

- The keyword **class** was introduced in ES6 more of a syntactic sugar for prototype-based inheritance
- If two objects inherit properties from the same prototype, then they become the instances of the same class
- **Object.create**() function creates a new object that inherits from the specified object

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Creating Classes

```
class Circle {
    constructor(radius) {
        this._radius = radius
    }
    get radius() {
        return this._radius
    }
    set radius(newRadius) {
        this._radius = newRadius
    }
}

const obj = new Circle(5)
console.log(obj.radius) // 5
obj.radius = 10
console.log(obj.radius) // 10
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Creating Classes

- It uses **class** keyword and the name usually starts with a capital letter
- The keyword **constructor** is used to initialize the variables
- If initialization is not needed, we can drop the **constructor** keyword
- An **empty constructor** will be created implicitly by the program

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Static Methods

- Class declarations are **not hoisted** like function declarations that is we cannot instantiate a class anywhere in the code but we need to define them before
- All code within the body of a class declaration is implicitly in strict mode, even if **no "use strict"** directive appears
- **Static methods** are methods that belong to the class itself, rather than to instances of the class

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Static Methods

- They are called on classes and not on objects created from those classes
- **Static methods** are useful for utility functions that do not depend on instance-specific data

```
class MathUtils {
    static add(a, b) {
        return a + b;
    }

    static subtract(a, b) {
        return a - b;
    }
}
```

```
// Output: 8
console.log(MathUtils.add(5, 3));
// Output: 2
console.log(MathUtils.subtract(5, 3));
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Adding Methods

- We can add methods to existing classes using the **prototype**
- It extends the functionality of built-in classes or custom classes without modifying their original source code
- Every function has a prototype property that is used to attach properties and methods to objects created by that function

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Adding Methods

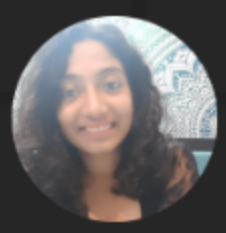
```
// Adding a new method 'sum' to the Array prototype
Array.prototype.sum = function() {
    return this.reduce((accumulator, currentValue)
    => accumulator + currentValue, 0);
};

// Using the new method
const numbers = [1, 2, 3, 4, 5];
console.log(numbers.sum()); // Output: 15
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Public Fields

- Public fields are accessible from **anywhere**, both inside and outside the class. They are declared directly within the class body

```
class Person {
    // Public field
    name = 'Unknown';
    constructor(name) {
        if (name) {
            this.name = name;}}
}

const person1 = new Person('Alice');
console.log(person1.name); // Output: Alice
```



Arpitha Rajeev
arpitha.rajeev37@gmail.com

Private Fields

- Private fields are only accessible **within the class** they are declared in.
- They are prefixed with a **# symbol**
- Attempting to access them outside the class results in a syntax error.

```
class Person {
    // Private field
    #age = 30;
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Private Fields

```
constructor(name, age) {
    this.name = name;
    if (age) {
        this.#age = age;
    }
}

getAge() {
    return this.#age;
}

setAge(newAge) {
    if (newAge > 0) {
        this.#age = newAge;
    }
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Private Fields

```
const person2 = new Person('Bob', 25);
console.log(person2.getAge()); // Output: 25
person2.setAge(26);
console.log(person2.getAge()); // Output: 26
// console.log(person2.#age); // SyntaxError
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Subclasses

- **Subclasses** are created from the **parent class** using the **extends** keyword
- The **super keyword** is used to call the constructor and methods of the parent class to enable code reuse
- This was again introduced in **ES6** as a key part of object-oriented programming

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Subclasses

```
// Parent Class
class Animal {
    constructor(name) {
        this.name = name;
    }
    speak() {
        console.log(`Hi ${this.name}.`);
    }
}
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Subclasses

```
// Subclass
class Dog extends Animal {
    constructor(name, breed) {
// Calls the constructor of the parent class
        super(name);
        this.breed = breed;
    }
// Calls the speak method of the parent class
    speak() {
        super.speak();
        console.log(`${this.name} barks.`);
    }
}
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Subclasses

```
const dog = new Dog('Rex', 'German Shepherd');
dog.speak();
```

- **Method Overriding**: Dog class overrides the speak method, calling the parent class's method using super.speak() and then adding additional behavior
- It prints **Hi rex, Rex barks**
- If we don’t use **super**() keyword before accessing **this**, it throws error

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Delegation over Inheritance

- **Composition:** Creating a class that includes instances of other classes and delegates behavior to these instances
- Reduces complexity by avoiding deep inheritance chains

```
class Engine {
    start() {
        console.log('Engine starts.');
    }
}
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Delegation over Inheritance

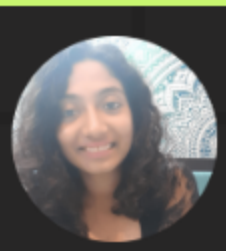
```
class Car {
    constructor(brand) {
        this.brand = brand;
        this.engine = new Engine();
    }

    start() {
        console.log(`${this.brand} car starts.`);
        this.engine.start();
    }
}
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Delegation over Inheritance

```
const car = new Car('Toyota');
car.start();
// Output:
// Toyota car starts.
// Engine starts.
```

- Car has an Engine, demonstrating a **"has-a" relationship**
- The Car class **delegates** the behavior of starting the engine to the Engine class instance.



Arpitha Rajeev
arpitha.rajeev37@gmail.com

Follow Me

For more such content on Software Development

Arpitha Rajeev
arpitha.rajeev37@gmail.com